# EE–559: Mini-project II

François Fleuret

https://fleuret.org/dlc/

[version of: May 1, 2018]

## 1 Introduction

The objective of this project is to design a mini "deep learning framework" using only pytorch's tensor operations and the standard math library, hence in particular **without using autograd or the neural-network modules.**

A zip file named `Proj2_StudentFamilyName1_StudentFamilyName2_StudentFamilyName3.zip` must be uploaded to the Moodle of the course before

**Friday May 18th, 23:59.**

It should contain, and only contain:

- the python source files, including a main executable `test.py` to call without arguments, and
- a 3–5 pages report in pdf.

The source code should be properly commented to facilitate its understanding.

**Exchange of code or report snippets between groups, and materials taken "from the web" are forbidden. Also, every student should have a clear understanding of her/his group's entire source code and report. This will be checked during the oral presentation.**

## 2 Objective

Your framework should import only `torch.FloatTensor` and `torch.LongTensor` from pytorch, and use no pre-existing neural-network python toolbox.

Your framework must provide the necessary tools to:

- build networks combining fully connected layers, Tanh, and ReLU,
- run the forward and backward passes,
- optimize parameters with SGD for MSE.

You must implement a test executable named `test.py` that imports your framework and

- Generates a training and a test set of $1,000$ points sampled uniformly in $[0,1]^2$, each with a label 0 if outside the disk of radius $1/\sqrt{2\pi}$ and 1 inside,
- builds a network with two input units, two output units, three hidden layers of 25 units,

- trains it with MSE, logging the loss,
- computes and prints the final train and the test errors.

# 3 Suggested structure

You are free to come with any new ideas you want, and grading will reward originality. The suggested simple structure is to define a class

```
class Module(object):

    def forward(self, *input):
        raise NotImplementedError

    def backward(self, *gradwrtoutput):
        raise NotImplementedError

    def param(self):
        return []
```

and to implement several modules and losses that inherit from it.

Each such module may have tensor parameters, in which case it should also have for each a similarly sized tensor gradient to accumulate the gradient during the back-pass, and

- `forward` should get for input, and returns, a tensor or a tuple of tensors.
- `backward` should get as input a tensor or a tuple of tensors containing the gradient of the loss with respect to the module's output, accumulate the gradient wrt the parameters, and return a tensor or a tuple of tensors containing the gradient of the loss wrt the module's input.
- `param` should return a list of pairs, each composed of a parameter tensor, and a gradient tensor of same size. This list should be empty for parameterless modules (e.g. ReLU).

Some modules may requires additional methods, and some modules may keep track of information from the forward pass to be used in the backward.

You should implement at least the modules `Linear` (fully connected layer), `ReLU`, `Tanh`, `Sequential` to combine several modules in basic sequential structure, and `LossMSE` to compute the MSE loss.